

Mikrokontroléry a embedded systémy

Hands-on: Mikrokontroléry STM32

Ing. Aleš POVALAČ, Ph.D.

povalac@vut.cz

Ústav radioelektroniky
Vysoké učení technické v Brně, FEKT

Školení pro SPŠ Třebíč

březen 2023

- Založte nový projekt v STM32CubeIDE přes File / New / STM32 Project / Board Selector / NUCLEO-F030R8. Budeme využívat HAL knihovny, proto ponechte Targeted Project Type na STM32Cube. Potvrďte inicializaci všech periférií do výchozího nastavení.
- Překlad (Build all) lze spustit pomocí zkratky Ctrl+B, ladění (tj. spuštění, Debug) projektu pomocí F11.
- Při prvním spuštění se zobrazí výběr debuggeru, volte STM32 Cortex-M C/C++ Application. Pokud je firmware debuggeru v připojené vývojové desce neaktuální, může být vyžádán upgrade.
- Během krokování ladění lze s výhodou používat klávesové zkratky, viz menu Run.
- Git repo: <https://github.com/alpov/SPST>

Rozblikání LED a loopback STM32F030

- Pin pro LD2 je ve výchozím stavu nastaven jako výstupní. Veškerou inicializaci zajišťuje kód vygenerovaný CubeMX.
 - Pro využití LED1 a LED2 na shieldu je třeba je nastavit (LED1=PA4, LED2=PB0)
- V souboru main.c doplňujte vlastní kód vždy mezi komentáře **USER CODE BEGIN** a **USER CODE END**. Pokud toto pravidlo porušíte, bude váš kód smazán při přegenerování projektu z CubeMX.
- Vývoj začněte funkcí main(). Do nekonečné smyčky vložte negaci příslušného pinu a krátké čekání.

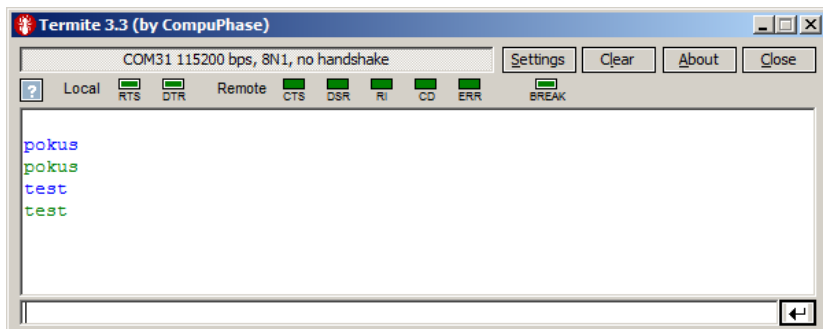
```
HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);  
HAL_Delay(200);
```

- Zařízením loopback se rozumí okamžité odeslání každého přijatého bajtu zpět odesílateli. Pro testování sériové komunikace je optimální program [Termite](#), výchozí rychlost portu je 38400 Bd.
- Funkci realizujeme pomocí HAL knihoven v nekonečné smyčce main()u:

```
uint8_t c;
```

```
HAL_UART_Receive(&huart2, &c, 1, HAL_MAX_DELAY);
```

```
HAL_UART_Transmit(&huart2, &c, 1, HAL_MAX_DELAY);
```



DMA, obsluha textového protokolu STM32F429

Kruhový buffer s DMA (1/4)

- Pro rozhraní USART3 nastavte DMA Request pro příjem, využijte kruhový režim (circular).

DMA Request	Stream	Direction	Priority
USART3_RX	DMA1 Stream 1	Peripheral To Memory	Low

DMA Request Settings

Peripheral		Memory
Mode	Circular	<input checked="" type="checkbox"/>
Increment Address	<input type="checkbox"/>	
Use Fifo	<input type="checkbox"/>	
Threshold		
Data Width	Byte	Byte
Burst Size		

- Přijem je nejvýhodnější realizovat s využitím kruhového bufferu. Jednotlivé přijímané znaky jsou postupně ukládány do bufferu, po dosažení jeho konce se automaticky pokračuje znovu od začátku. Zápisový index lze tedy vypočítat na základě délky bufferu a registrů DMA přenosu, čtecí index se obsluhuje softwarově.

- Deklarujeme buffer a indexy čtení a zápisu:

```
#define RX_BUFFER_LEN 64
static uint8_t uart_rx_buf[RX_BUFFER_LEN];
static volatile uint16_t uart_rx_read_ptr = 0;
#define uart_rx_write_ptr (RX_BUFFER_LEN - hdma_usart3_rx.Instance->NDTR)
```

- Nejdříve je třeba aktivovat DMA čtení z UARTu (sekce USER CODE 2):

```
HAL_UART_Receive_DMA(&huart3, uart_rx_buf, RX_BUFFER_LEN);
```

- V hlavní smyčce kódu pak postupně zpracováváme jednotlivé bajty z bufferu. Dojde-li k zablokování kódu po určitou dobu (např. vlivem zpracování ISR), data se „nahromadí“ v bufferu a zpracují se, jakmile je k dispozici procesorový čas:

```
while (uart_rx_read_ptr != uart_rx_write_ptr) {
    uint8_t b = uart_rx_buf[uart_rx_read_ptr];
    if (++uart_rx_read_ptr >= RX_BUFFER_LEN) uart_rx_read_ptr = 0;
                                                // increase read pointer
    uart_byte_available(b); // process received byte with the RX state machine
}
```


- Funkce `uart_byte_available()` ukládá tisknutelné znaky do dalšího bufferu, který již obsahuje poskládaný textový příkaz. Po nalezení konce řádku zavolá obsluhu pro vyhodnocení příkazu a výpis výstupu.
- Vhodnou velikost bufferu pro textový příkaz `CMD_BUFFER_LEN` je třeba definovat (např. 256B).

```
static void uart_byte_available(uint8_t c)
{
    static uint16_t cnt;
    static char data[CMD_BUFFER_LEN], resp[CMD_BUFFER_LEN];

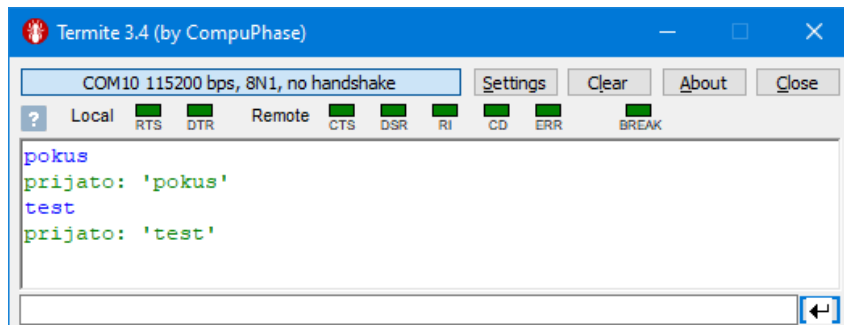
    if (cnt < CMD_BUFFER_LEN && c >= 32 && c <= 126) data[cnt++] = c;
    if ((c == '\n' || c == '\r') && cnt > 0) {
        data[cnt] = '\0';
        process_command(data, resp);
        HAL_UART_Transmit(&uart3, (uint8_t*)(resp), strlen(resp), HAL_MAX_DELAY);
        cnt = 0;
    }
}
```

Kruhový buffer s DMA (4/4)

- Do funkce **process_command()** je pro otestování vhodné doplnit zpětný výpis přijatého příkazu:

```
void process_command(char *cmd, char *resp)
{
    sprintf(resp, "prijato: '%s'\n", cmd);
}
```

- Aby fungovalo formátování pomocí **sprintf()**, je třeba inkludovat **<stdio.h>**. Výchozí rychlost portu na NUCLEO-F429ZI je 115200 Bd.



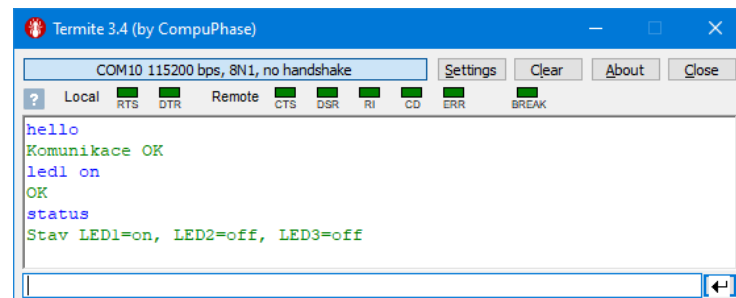
- Implementuje příkazy HELLO, LED1, LED2, LED3 a STATUS.
 - Příkaz HELLO vypíše uvítací text.
 - Příkazy LED1 a LED2 rozsvítí nebo zhasnou jednotlivé LED kontrolky podle údaje ON nebo OFF.
 - Příkaz STATUS vrátí stav obou LED.
- Použijte parsování příkazů pomocí funkce **strtok()** <http://www.cplusplus.com/reference/cstring/strtok/>, oddělovačem bude znak mezery, inkludujte hlavičkový soubor **<string.h>**. Jednotlivé příkazy rozlišujte se zanedbáním velikosti znaků pomocí funkce **strcasecmp()** <http://www.cplusplus.com/reference/cstring/strcasecmp/>.
- Využijte funkcí **HAL_GPIO_WritePin()**, **HAL_GPIO_ReadPin()** a formátovaný výpis. Opakovaná volání **strtok()** s prvním parametrem **NULL** vrací další tokeny ve formě C řetězců.

Obsluha textového protokolu (2/2)

```
char *token;
token = strtok(cmd, " ");

if (strcasecmp(token, "HELLO") == 0) {
    sprintf(resp, "Komunikace OK\n");
}
else if (strcasecmp(token, "LED1") == 0) {
    token = strtok(NULL, " ");
    if (strcasecmp(token, "ON") == 0) HAL_GPIO_WritePin(LD1_GPIO_Port, LD1_Pin, GPIO_PIN_SET);
    else if (strcasecmp(token, "OFF") == 0) HAL_GPIO_WritePin(LD1_GPIO_Port, LD1_Pin, GPIO_PIN_RESET);
    sprintf(resp, "OK\n");
}
else if (strcasecmp(token, "LED2") == 0) {
    // ...
}
// ...
else if (strcasecmp(token, "STATUS") == 0) {
    bool ld1 = HAL_GPIO_ReadPin(LD1_GPIO_Port, LD1_Pin);
    bool ld2 = HAL_GPIO_ReadPin(LD2_GPIO_Port, LD2_Pin);
    bool ld3 = HAL_GPIO_ReadPin(LD3_GPIO_Port, LD3_Pin);

    sprintf(resp, "Stav LED1=%s, LED2=%s, LED3=%s\n", ld1?"on":"off", ld2?"on":"off", ld3?"on":"off");
}
else {
    sprintf(resp, "Neznamy prikaz\n");
}
```



FreeRTOS, Ethernet STM32F429

- Budeme využívat knihovnu lwIP pod systémem FreeRTOS. V CubeMX je třeba povolit FreeRTOS (CMSIS_V1) a podle doporučení generátoru přepnout časovou základnu (SYS / Timebase Source = TIM14) a zapnout reentranci newlibu (FreeRTOS / Advanced Settings / USE_NEWLIB_REENTRANT = Enabled).
- Vzhledem k rozsahu projektu je potřeba zvětšit velikost haldy FreeRTOS (FREERTOS / Config Parameters / Memory management settings / TOTAL_HEAP_SIZE = 32768) a zásobníku výchozí úlohy (defaultTask / Stack Size = 1024).
- Veškeré úlohy pro Ethernet budou vytvářeny dynamicky, v konfiguraci FreeRTOS pod CubeMX je však potřeba vytvořit úlohu pro obsluhu USART z předchozích kroků, pokud chceme zachovat její funkci.

Tasks								
Task Name	Priority	Stack S...	Entry Function	Code Ge...	Parameter	Allocation	Buffer Name	Control Bloc...
defaultTask	osPriorityNormal	1024	StartDefaultTask	Default	NULL	Dynamic	NULL	NULL
SerialTask	osPriorityIdle	128	StartSerialTask	Default	NULL	Dynamic	NULL	NULL

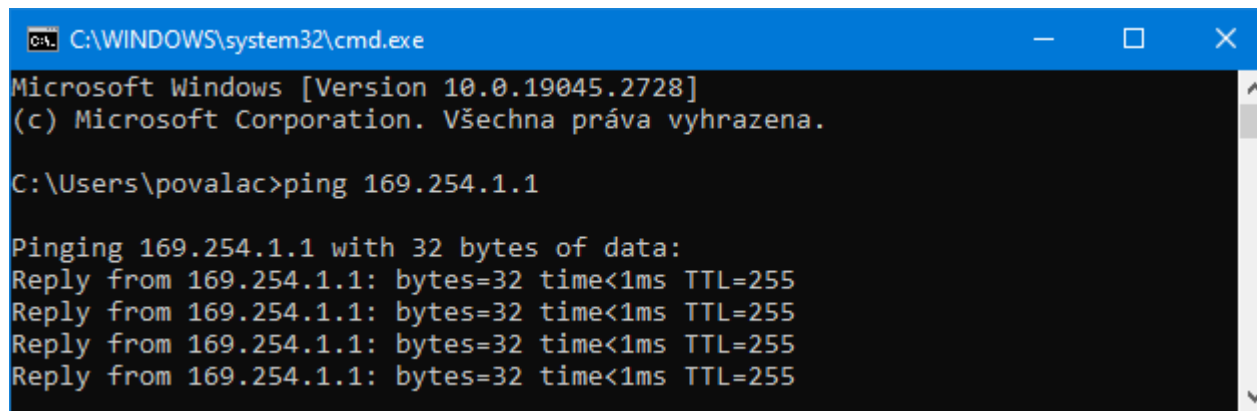
- Úloha bude obsahovat inicializaci DMA přenosu a původní nekonečnou smyčku z main().

```
void StartSerialTask(void const *argument)
{
    /* USER CODE BEGIN StartSerialTask */
    HAL_UART_Receive_DMA(&huart3, uart_rx_buf, RX_BUFFER_LEN);

    /* Infinite loop */
    for (;;) {
        while (uart_rx_read_ptr != uart_rx_write_ptr) {
            uint8_t b = uart_rx_buf[uart_rx_read_ptr];
            if (++uart_rx_read_ptr >= RX_BUFFER_LEN)
                uart_rx_read_ptr = 0; // increase read pointer

            uart_byte_available(b); // process received byte
        }
        osDelay(1);
    }
    /* USER CODE END StartSerialTask */
}
```

- Aktivujte lwIP (Enabled), v Platform Settings zvolte LAN8742 a příslušnou solution pro správné BSP.
- Pro vývojovou desku využijeme link-local adresu 169.254.1.1/16, v General settings tedy deaktivujte LWIP_DHCP a nastavte adresu 169.254.1.1, masku 255.255.0.0.
- Po připojení k počítači se rozsvítí LED kontrolky na ethernetovém konektoru, ověřte funkčnost pomocí příkazu ping na adresu 169.254.1.1.



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.19045.2728]
(c) Microsoft Corporation. Všechna práva vyhrazena.

C:\Users\povalac>ping 169.254.1.1

Pinging 169.254.1.1 with 32 bytes of data:
Reply from 169.254.1.1: bytes=32 time<1ms TTL=255
Reply from 169.254.1.1: bytes=32 time<1ms TTL=255
Reply from 169.254.1.1: bytes=32 time<1ms TTL=255
Reply from 169.254.1.1: bytes=32 time<1ms TTL=255
```


- Příklad tcpecho je součástí balíku lwIP. Poslouchá na TCP portu 7 a veškerá přijatá data odesílá zpátky. Použitá verze je postavena na netconn API.
- Soubor tcpecho.c najdete v např. repozitáři (%USERPROFILE%\STM32Cube\Repository\STM32Cube_FW_F4_V1.27.1\Projects\STM32469I_EVAL\Applications\LwIP\LwIP_UDPTCP_Echo_Server_Netconn_RTOS\Src), okopírujte ho mezi zdrojové kódy projektu (Core\Src). V souboru main.c deklarujte prototyp inicializační funkce:

```
void tcpecho_init(void);
```

- Tuto funkci zavolejte v rámci inicializace (tj. před nekonečnou smyčkou) ve StartDefaultTask:

```
/* Initialize tcp echo server */  
tcpecho_init();
```

- Ověřte pomocí PuTTY připojení na port 7, veškeré odeslané zprávy bude tcpecho server vracet zpět. Používejte Connection type nastavený na Raw.

- Příklad tcpecho modifikujeme pro obsluhu textového protokolu. Data místo posílání zpět (netconn_write()) budeme znak po znaku předávat funkci pro zpracování:

```
while (len--) telnet_byte_available(*(uint8_t*)data++, newconn);
```

- Zpracování pak bude volat stejný parser jako UART komunikace:

```
#include <string.h>
#define CMD_BUFFER_LEN 256

void process_command(char *cmd, char *resp);

static void telnet_byte_available(uint8_t c, struct netconn *conn)
{
    static uint16_t cnt;
    static char data[CMD_BUFFER_LEN], resp[CMD_BUFFER_LEN];

    if (cnt < CMD_BUFFER_LEN && c >= 32 && c <= 127) data[cnt++] = c;
    if ((c == '\n' || c == '\r') && cnt > 0) {
        data[cnt] = '\0';
        process_command(data, resp);
        netconn_write(conn, resp, strlen(resp), NETCONN_COPY);
        cnt = 0;
    }
}
```

- V konfiguraci lwIP aktivujte HTTP server (LWIP / HTTPD / LWIP_HTTPD = Enabled).
- Tento poměrně komplexní HTTP server je postavený na raw API. Webové stránky v nejjednodušším případě čte z paměti programu, kam jsou vloženy pomocí souboru `fsdata_custom.c`.
- Soubor `fsdata_custom.c` se vytváří pomocí programu `makefsdata.exe` (zdrojový kód je součástí balíku lwIP). Ten zkompiluje HTML soubory a obrázky ve složce `Fs` do souboru `fsdata.c`.
- Pro jednoduchost využijeme předpřipravený `fsdata_custom.c`, který můžeme najít v repozitáři např. ve složce `%USERPROFILE%\STM32Cube\Repository\STM32Cube_FW_F4_V1.27.1\Projects\STM32F429ZI-Nucleo\Applications\LwIP\LwIP_HTTP_Server_Netconn_RTOS\Src`. Tento soubor nakopírujte mezi **hlavičkové** soubory projektu (`Core/Inc`) a vyřadte z kompilovaných souborů (pravé tlačítko pro kontextové menu / `Resource Configurations / Exclude from Build`).
 - Tento postup řeší situaci, kdy soubor chceme mít přístupný v IDE, ale nekompilujeme ho samostatně – je totiž pomocí `#include` vložený do souboru `Middlewares/Third_Party/LwIP/src/apps/http/fs.c`. Musí tedy být v prohledávané cestě, aby fungoval `#include` (proto `Core/Inc`), a zároveň nesmí být samostatně kompilován (jinak by symboly v něm uvedené byly definované vícenásobně).

Backup: HTTP server (2/2)

- V souboru main.c includujte příslušný hlavičkový soubor:
#include "lwip/apps/httpd.h"

- A ve StartDefaultTasku inicializujte server:

```
/* Initialize HTTP server */  
httpd_init();
```

- Ověřte funkci pomocí webového prohlížeče na adrese
<http://169.254.1.1/STM32F4xx.html>

